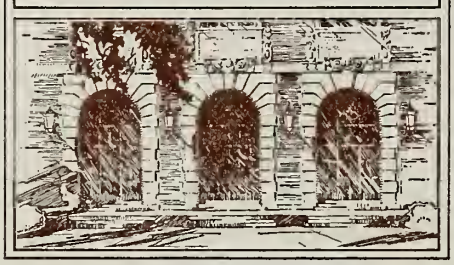


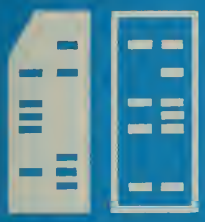
UIUCDCS-R-74-626

THE GUIDE
AN INFORMATION SYSTEM

by

Jean Louis Pradels

March 1974



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-74-626

THE GUIDE
AN INFORMATION SYSTEM

by

Jean Louis Pradels

March 1974

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois

This work was supported in part by the National Science Foundation under Grant No. US NSF GJ-31222 and NSF EC 41511 and was submitted in partial fulfillment for the Doctor of Philosophy degree in Computer Science, 1974.

THE GUIDE
AN INFORMATION SYSTEM

Jean Louis Pradels, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1974

This thesis describes the design and implementation of a conversational information system, called the Guide. The environment in which it must work is sufficiently different from that of other systems which have similar goals, that new techniques had to be devised to make its realization possible. This thesis also shows that these techniques are applicable in contexts other than ours, and would probably lead to improved performance in a number of information systems based on other approaches.

ACKNOWLEDGMENT

The author wishes to express his sincere gratitude and appreciation to his supervisor, Professor Jurg Nievergelt, for suggesting this research, for his guidance, stimulating discussions, and encouragements.

He would like to thank Messrs. Ron Danielson, Dave Eland, Dave Embley, Prabhaker Mateti and many other students for their time spent in using the Guide. The feedback received has been essential for its improvement.

He is thankful to Dave Eland for reading and commenting on the manuscript and for discussions on the project.

Finally, he would like to express his gratitude to the Department of Computer Science of the University of Illinois for its financial support and to the offset department for the reproduction of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

1. PRESENTATION OF THE GUIDE

1.1 Introduction

This thesis describes the design and implementation of a conversational information system, called the Guide. The environment in which it must work is sufficiently different from that of other systems which have similar goals, that new techniques had to be devised to make its realization possible. This thesis also shows that these techniques are applicable in contexts other than ours, and would probably lead to improved performance in a number of information systems based on other approaches.

Let us first describe the environment in which the Guide must work.

The Department of Computer Science at the University of Illinois is involved in a project to develop an automatic instructional system capable of teaching introductory computer science courses [8]. This instructional system is being implemented on PLATO IV, a computer-assisted-instruction system developed by the Computer-Based Education Research Laboratory at this university [1].

The computer science instructional system should be useful as a supplement to classroom instruction in a variety of introductory computer science courses, and should also be

usable without any instructor at all by anybody having access to a PLATO terminal who wants to learn on his own some particular aspect of computer science. For these reasons the instructional system is organized as a collection of relatively independent and self-contained components. This collection is expected to keep growing over the entire lifetime of the system.

The purpose of the Guide is to give students and occasional users convenient access to this large body of encyclopedic information. The Guide is to be an automated version of a reference librarian as well as an academic advisor. It has to answer questions about what the collection contains on a certain topic, and give advice on what lesson a student might take, based on his goals and past performance. These functions of the Guide had to be realized under very stringent constraints on memory space and processing time.

The type of requests to be handled convinced us that a natural language communication between the users and the computer was an extremely desirable feature. Although natural language is not always the best medium for communicating with a computer, it is suitable whenever the concepts carried by sentences cannot be easily expressed by pressing keys and when the system is used by a large number of people who are occasional users.

This type of interaction is not new. Baseball [7] has been the most successful of the first generation of Questions Answering Systems. Deacon [5], Airline Guide [14],

Lsnlis [15] are among recent remarkable realizations and represent a variety of philosophies and ideas which enter into the design of these systems. However, a natural language communication had never been attempted under the stringent constraints imposed by PLATO:

- The component which "understands" requests formulated in fairly unrestricted English, had to use an area of memory less than 4K words long for the dictionary of English words and grammatical information.
- This program moreover could utilize the central processor only a few milliseconds each second (because of time-sharing considerations), and had to respond to the user within at most ten seconds.

A survey of similar systems, interacting with users in natural language, revealed that previous approaches were unfeasible in our situation. The interpretation of requests was done by large and time consuming programs which could not meet the stringency of our requirements. Hence, we were faced with the challenging research problem of finding a new approach which would fit our constraints and at the same time make the system perform as well as others. The technique that we have developed meets those requirements and is not only computationally efficient but is also theoretically interesting.

This thesis is mainly concerned with the structural organization of this information system, the detailed definition

of its components, and the research and implementation of a new technique which allows a natural language interface with its users. The implementation of retrieving routines has been left as a subsequent task.

The system is written in a language called TUTOR, a FORTRAN-like high level language designed for CAI. However, the specific CAI features of TUTOR have not been used, and the implementation of a similar system, using the same techniques, in another high level language should be straightforward.

1.2 Essence of Previous Approaches

We began this research by selecting several existing information systems which represented the wide variety of approaches which had already been taken. The method generally used to interpret requests expressed in a natural language consists of a syntactic-semantic analysis. Each particular approach differs in the importance given to the syntax analysis versus the semantic analysis, in the dependence of these two phases, and in the techniques which are used. Their common feature is that an important set of grammatical rules needs to be stored, the size of which is related to the fluency of the system. (Deacon [5], for example, needs more than 900 such rules.) The previous attempts to interpret natural language sentences have also followed the general scheme of applying a general language processing technique to a limited domain of discourse.

1.3 Essence of Our Approach

Since our memory limitations could not allow us to store a large amount of information relative to the English grammar, we took a different approach. It consists of first analyzing the properties of the domain of discourse, and then designing a suitable method to process sentences in this specific domain. This approach, however, is general enough to be used for other domains of discourse (see Chapter 4). We give a detailed description of the method in §2.4, but let us briefly outline here the underlying philosophy of the method and the advantages it represents in regard to the problems we faced.

We have characterized the domain of discourse by a tree structure in which a node indicates a partial information about what a sentence of the domain might be about. The interpretation of a request consists then essentially of searching in this structure for a sequence of nodes reflecting the meaning of the request.

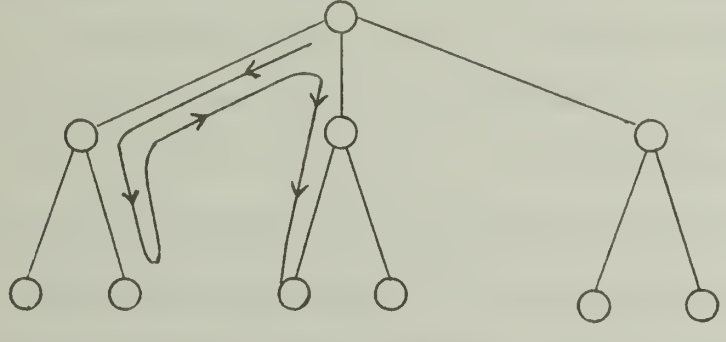


Figure 1. Heuristic Search for an Interpretation

The ordering of these various nodes reflects also the general syntactic properties of the sentences of the domain of discourse. The search for the meaning of a sentence is similar to the thought process of a person reading a sentence having context dependent words. This search is done by a non-deterministic automaton which analyzes the sentence from left to right and has this tree structure as its state diagram. It then follows that syntactic and semantic analysis are two mixed phases which permit a partial understanding as the request is processed.

The approach has several desirable properties:

- It does not require storing a grammar of English in the conventional sense, and hence allows a substantial saving of memory space.
- It permits better response to requests slightly outside the acceptable set because of the concept of partial understanding which permits the system to generate comments telling why a request is not accepted and how it should be rephrased.
- It permits commonly used ungrammatical requests to be accepted.
- Requests are processed by computationally efficient search which allows a fast interpretation.
- It can be used for many other domains of discourse.

1.4 Achievement of Initial Goal

The first experiments with the Guide show that our initial goal has been achieved:

- The system fits the PLATO constraints in regard to memory size and processor time allocation.
- It has comparable performance with other similar systems in regard to fluency and completeness.
- Requests are processed much faster than they are in other systems with a comparable domain of discourse.

Fluency. A system is said to be fluent if it accepts different forms of the same request. Because of our approach, fluency does not depend on how many grammar rules we have in memory, but on the size of the vocabulary data-bases and the complexity of the state diagram of the automaton. Many different forms of the same request are accepted, grammatically correct or not.

Completeness. A system is logically complete if it can process any relevant request. In order to do so, two conditions must be satisfied: The formal intermediate language must be complete itself and the translator must be able to map any relevant requests into the corresponding formal representation. This last condition is the most difficult and no program has so far achieved this goal. We took a pragmatic approach in this respect, consisting of designing a formal language in which the most relevant requests could be expressed. Other requests generate comments which express the system's limitations.

Processing time of a request. The most common requests consist of 10 to 15 words and are translated using on the average 60 milliseconds of processor time on a CDC Cyber 70 computer.

1.5 Comparison With Other Similar Systems

- In regard to interaction with the user, the main limitation of our system comes from the fact that it does not memorize previous requests. Hence it does not accept general anaphoric sentences. However, the method can, in principle, be extended to make possible a dialog with the user. The constraints we faced prevented us from doing so.
- In regard to performance and complexity, since our approach is simpler, a natural and immediate question is: Can the same method be used to process requests handled by other similar systems? It appears that our approach is suitable when the number of different varieties of sentence syntax is not too high, since the state diagram of the automaton describes all possible general syntax forms--requests for information satisfy this condition. The most frequently used syntax is the one of a request where the user indicates first what he wants to

know and then gives specifications further describing his domain of interest. We will show in Chapter 5 how the same method can be used to process requests about the Official Airline Guide (74) of Woods.

- In regard to processing time, requests are "understood" much more rapidly. This feature makes the method suitable for real time systems where short response time is necessary.

2. DESCRIPTION OF THE GUIDE

2.1 Overall Description

The flow diagram in Figure 2 gives an overall description of the system and of its various components. This structure is frequently encountered among similar systems.

A request in English is first translated into a formal language, which is an interface between the retrieving components and the users. This mapping represents the "understanding" function of the system. Once a request has been accepted and mapped into this intermediate language, another component, the request processor, takes it as an input and uses the information stored in the four data bases to give back an appropriate answer. Its output is then handed out to a third component which displays the answer in current English. This second translation process is much less complex than the first one. It consists mainly of selecting some stored English sentences and merging them eventually with the retrieved information.

Most of the similar systems can answer only if the requests are relevant to the data bases. Questions which do not belong to the accepted set, even slightly outside, are rejected as not understandable. This restriction limits seriously the flexibility and usefulness of such systems. It confines the user within uncomfortable and sometimes frustrating

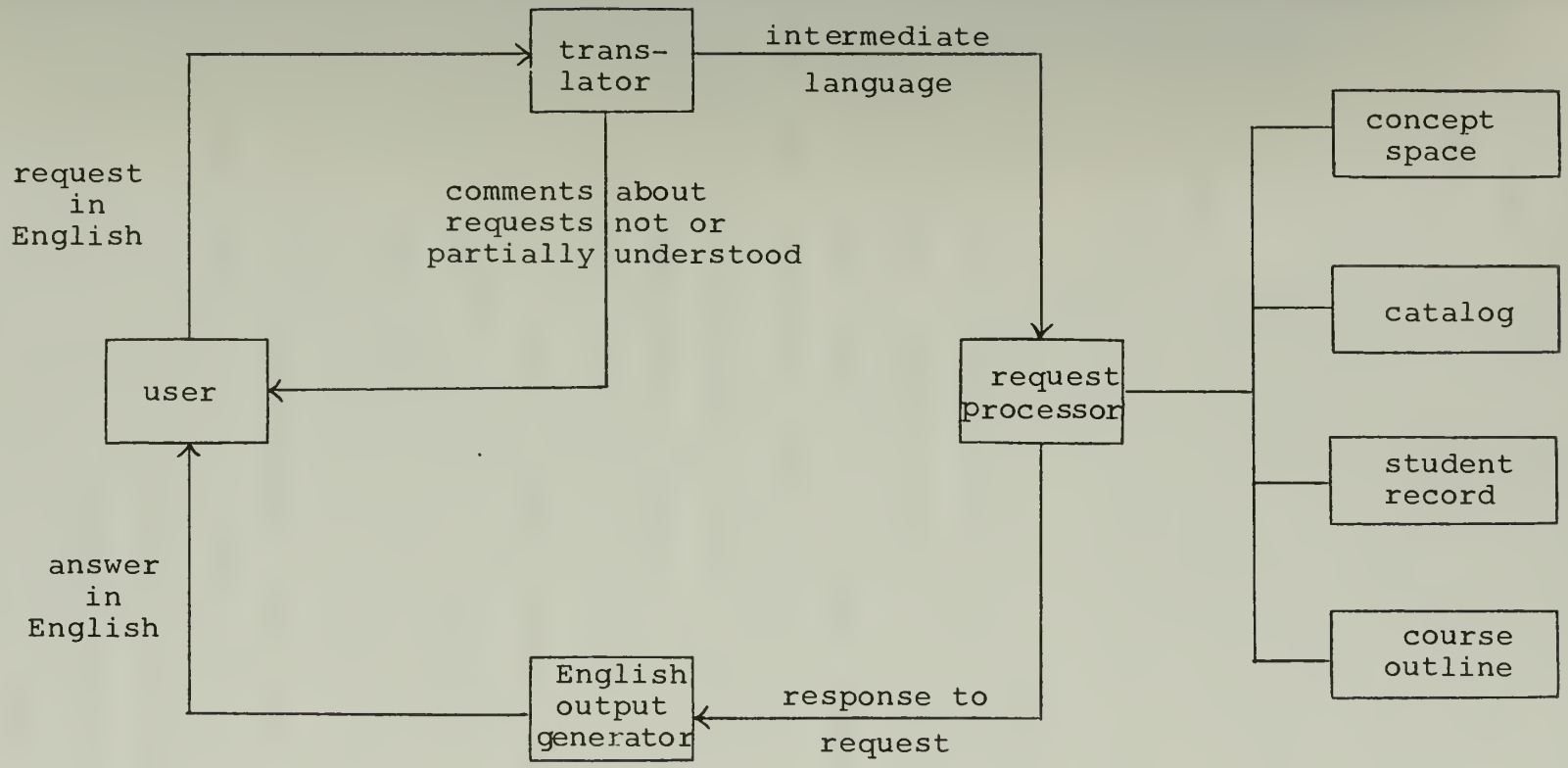


Figure 2. Overall Description of the Guide

boundaries. The concept of partial understanding allows the system a better interaction with the user. This is represented on the flow diagram by the feedback going from the translator to the user. The nature of a comment is a function of the distance of the rejected request from the accepted set and tells the user what to do in order to rephrase his sentence in a better way. This "partial understanding" feature makes the system cooperate with the user to achieve understanding.

2.2 The Domain of Discourse

We have described in the first chapter the purpose of the system. Its objective is to guide students through the library of computer sciences lessons, according to their goals and past performance. Users get selective access to the information they need by means of a request typed in English. These requests fall broadly into several categories:

- Requests for lesson names given specifications like keywords, authors, date, ...

  "Give me a lesson about Fortran and numerical analysis."

  "What PL/1 lessons should I take before October 10?"

- Requests for intrinsic information attached to each lesson, besides their names, like abstract, author, length, level of difficulty, ...

  "Who wrote lesson Somaga?"

  "Tell me about my next lesson."

- Requests for performance achieved by the student.

  "What grade did I get in the exam Fortxl?"

  "How much time did I spend in Fortif?"

- Requests for requirements imposed by an instructor for students in a course.

  "When should I take my next test on Fortran?"

  "What grade should I get in Fortxl?"

- Requests for counseling.

  "What lessons should I review before my next exam?"

  "Is Fortdo a better lesson than Fortif?"

We have illustrated these categories by listing a few requests. Many more will be listed in the following sections (sections 2.5, 3.4 and 4.1 in particular). A short inspection reveals that the domain is quite large and that sentences can be phrased using past, present or future tenses.

When we initially fixed the boundaries of the set of "acceptable" requests, we voluntarily discarded the last category because it appeared that in most cases the system could not have enough information to process them and algorithms would be too time consuming. However, we did decide to implement counseling routines to be activated periodically or at some critical points. These routines verify that a student properly follows the class schedule, meets the requirements of a course in which he is registered, and occasionally advises him which lessons to review for his own sake. In short, these

routines allow the Guide to assist a student in following a satisfactory program of study.

2.3 The Data Bases

When a request has been fully interpreted and accepted by the translator, its intermediate representation is handed out to another component, the request processor, which uses the information stored in several data bases to process the request and give an answer back to the user. Four data bases and one inverted file have been found necessary for this purpose. Let us now describe the need for these data bases by means of examples.

The concept space

Let us first look at the following two requests and answers:

student: "Show me what you have on bubble sort."

Guide: "The following lessons may be relevant:

lesson name: sorting
abstract: Introduction to sorting theory and algorithms.

lesson name: Files
abstract: Survey of file organization and file processing techniques."

student: "Do you have a lesson about recursion using Fortran?"

Guide: "I do not have such a lesson. Recursion is not a feature of Fortran."

In the first request, the Guide did not find a lesson having "bubble sort" as a keyword; however, it has expanded the request searching then for lessons having keywords closely related to "bubble sort."

In the second request, the request processor has searched the library of lessons looking for one having recursion and Fortran as keywords. None was found. However, the system has also given an additional comment: "Recursion is not a feature of Fortran."

For these two requests, the underlying operations, which have allowed the system to expand the first request and give additional comment to the second one, have been made possible by having stored in memory an arrangement of concepts specific to computer science, which cover all the topics treated in any of the courses. This arrangement consists of a classification scheme which links various concepts together by means of semantic relations.

Different models for the classification of concepts have been proposed. At one extreme there is Quillian's model [9] used by several deductive systems and which represents semantic information by general recursive graph structures. Its semantic map contains loops and different kinds of linkages. Simpler models can be found in the thesaurus of document retrieving systems. A thesaurus groups words together according to the subject concept to which they are related and concepts are organized on a hierarchical basis. This organization is

similar to the Dewey Decimal Classification (DDC) which is one of the best known and most frequently used library classification systems in the United States.

The arrangement which seemed the most suitable for the Guide lies between these two extremes. It consists of a graph where the nodes are sets of intrinsically similar concepts, and the links between the nodes are semantic relations like: "x is a predecessor of y," "x is an important concept of y," "x is not a concept of y ..." To each node is attached a level indicating the position of the concept in the underlying hierarchy.

The portion of the concept space in Figure 3 illustrates how the Guide has been able to expand the first request.

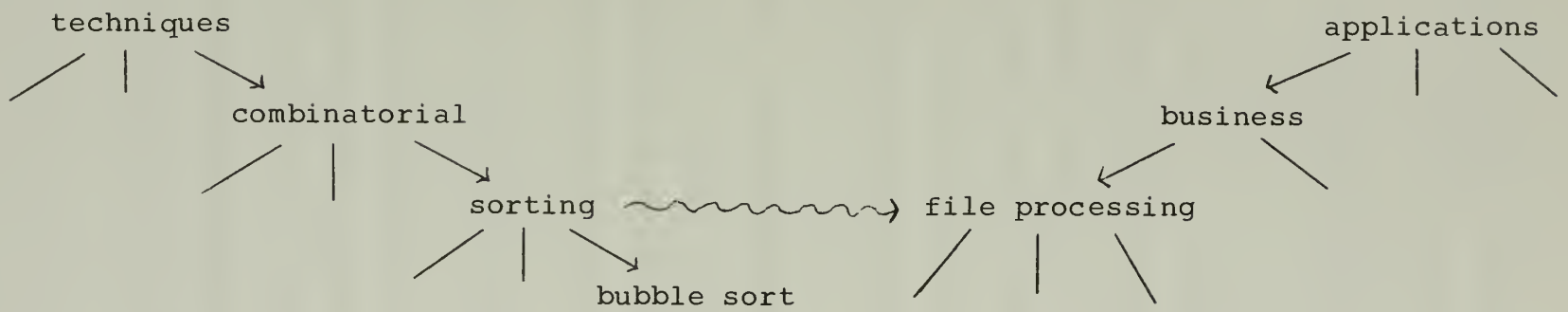
The concepts are represented in this data base by identifiers which have the same length and occupy 10 characters (60 bits). The full length word is stored in another data base which is used by the translator to recognize English words from a request. The correspondence between the identifier and the full length word is best illustrated by a few examples:

keyword up to 9 characters:	Identifier (10 characters)
FORTRAN	F O R T R A N _ _ (7)

keyword up to 19 characters:

NUME|RICAL |ANAL|YSIS — N U M E A N A L _ (18)

4 10 14



Types of relations

x ⟶ y : y is a subfield of x

x ⇝ y : x is a relevant for y

Figure 3. Fragment of the Concept Space

keyword up to 29 characters

FIXED POINT REPRESENTATION F I X T _ R T A T (26)

3 10 13 20 23

The binary value of the last character indicates the length of the keyword.

The catalog

This data base represents the library of computer sciences lessons. It can be seen as a graph where

- a node represents a lesson of the library and some inherent information related to it
- arcs between nodes express relations between lessons.

Let us look at the following requests to see the nature of information and relation we need to store.

R: "What is the abstract of lesson Fortif?"

R: "Give me an advanced Fortran lesson written by Smith."

R: "Do you have any lessons on Fortran and numerical analysis?"

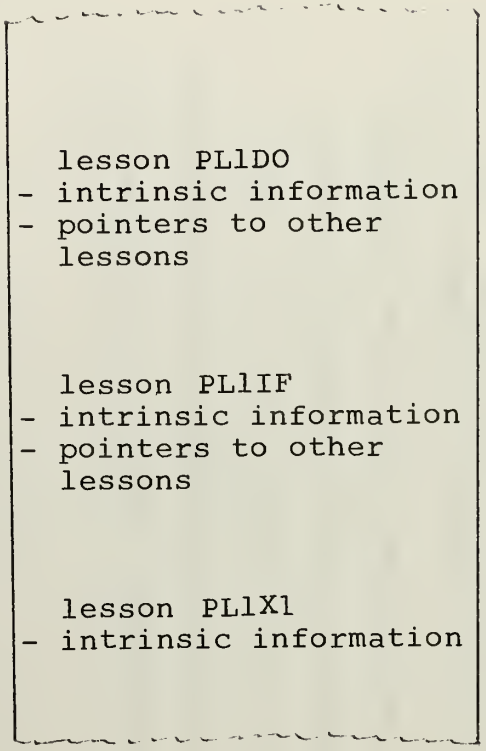
R: "Can I take lesson Fortdo before fortio?"

The information stored in each node indicates the name of a lesson, its type (lesson, exam, test), its length, author, level of difficulty, abstract, and a set of weighted keywords which allow the retrieval of lessons with high precision and relevance.

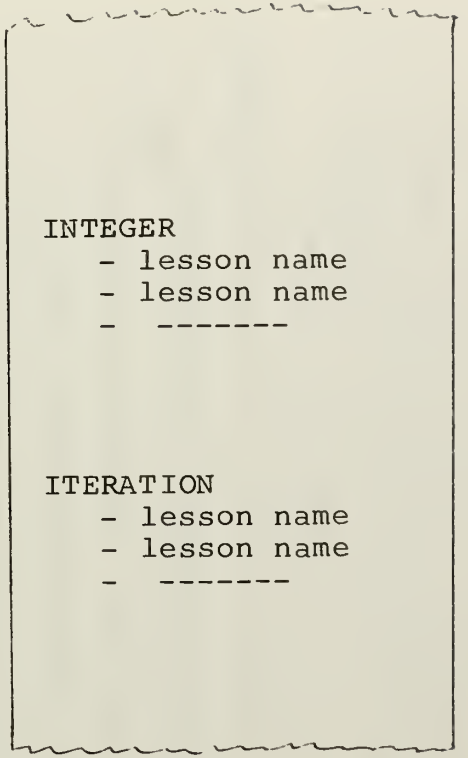
The relations between lessons are set by the manager of the library of lessons and indicate such relations as:

"x should be taken before y," "x can be used to practice for y," ... These relations impose an order on the library of lessons which is useful primarily for casual users and for students who are not registered in a course and who learn by themselves. A second level of similar relations is set in another data base which we will describe later and which reflects the educational strategy of the instructors of courses.

The organization of the catalog influences strongly the performance of the system since the retrieving of lesson information, given some of its attributes, is a frequent operation. A sequential file of lessons ordered alphabetically according to their names, where the attributes of each record indicate information inherent to the corresponding lesson, allows access to these records by a simple binary search whenever the name of a lesson is given. However, this structure would be prohibitive to retrieve lessons, given a list of keywords, since the number of operations involved would be considerable. It then becomes necessary to include an inverted file where the attributes of each record are a computer science keyword and a list of all the lessons having such a keyword. Several inverted files could be built, as many as the number of attributes of the records of our first file. But these files would be used less frequently and would use much memory space, which is our main constraint. Figure 4 describes these two files.



a) sequential file - sequence is on one attribute: the lesson name



b) inverted file - built from previous one by changing the sequence of records--the sequence is now on the keyword attribute.

Figure 4. Organization of the Catalog

The student record

In order to be able to answer requests like:

"What grade did I get in my last test?"

"How much time did I spend in Fortxl?"

"How many lessons have I taken so far?"

"What is the number of times I took lesson Fortdo?"

another data base is needed which indicates the path followed by each student through the library of lessons and the performance that has been achieved. Part of this information is stored automatically by the PLATO System when a student is taking a lesson.

The organization of this data base can be seen as a linear list of nodes, one list for each student, where a node indicates the name of a lesson taken, the student's performance, when the lesson has been taken, and the length of time spent on it.

The information stored in this data base can be retrieved at the demand of the student or of the instructor of the course in which the student is registered. This information is very useful for providing statistics to evaluate lessons.

The course outline

This last data base is used to allow the instructor of a course to specify a schedule and various requirements. The type of information which is needed is best seen by looking at a few requests:

"What grade should I get in the exam Fortxl?"

"What lessons should I learn before September 10?"

"Is Fortdo a prerequisite of Fortif?"

"When is my next exam?"

"How much time should I spend in Fortdo?"

"How many lessons are required in the course cs 101?"

The number of lessons used in a course is usually small, typically less than 20. Hence the organization of the data has little influence on the overall performance. It consists of a directed graph where the nodes are the lessons used in the course along with the performance expected in the lessons, and proper paths through the graph correspond to the sequence in which lessons should be taken. It is well known that many topics can be taught in different ways. One instructor may prefer to start teaching Fortran by discussing I/O statements first, for example, while another one adopts a different strategy. It is, therefore, essential to allow an instructor to arrange the set of lessons corresponding to his course, i.e., to allow him to specify his own teaching strategy. The relations which the instructor enters into the course outline are intended to complement those in the Catalog. They may even contradict them if an instructor feels that a specific part of the Catalog was not designed properly. The main difference is that the course outline is intended for a specific audience, while the Catalog is intended for all users. The course outline might be updated at the beginning of each semester by an instructor

who modifies his strategy by looking at statistics obtained from the student record data base, or by a new instructor. Figure 5 illustrates part of the structure of this data base.

2.4 Method for the "Understanding" of Requests

We started the search for a method by first analyzing the properties of the domain of discourse, in order to tailor the method to the real needs. We have outlined in section 1.3 the underlying philosophy of the method. Let us now describe it in more detail.

One major feature is that the sentences we deal with are requests; and requests, independently of their domain, have in general a simple syntax. If we denote by W what a person wants to know and by S the specifications limiting the domain of interest, we see that a number of requests have the general structure W.S:

"Let me see the abstract of lesson Fortdo."

"What is the location of the ship sea wolf?" [5]

"What is the departure time of flight AA-57 from Chicago?" [14]

The last sentence, for example, breaks into two parts:

"What is the departure time"

"flight AA-57 from Chicago"

The syntax, of course, is not always as simple as that, but it is essential to note that the number of different syntactic structures of requests is small. The parsing of a sentence, as described previously, is not defined in a very precise manner. The boundary between the W and S parts for

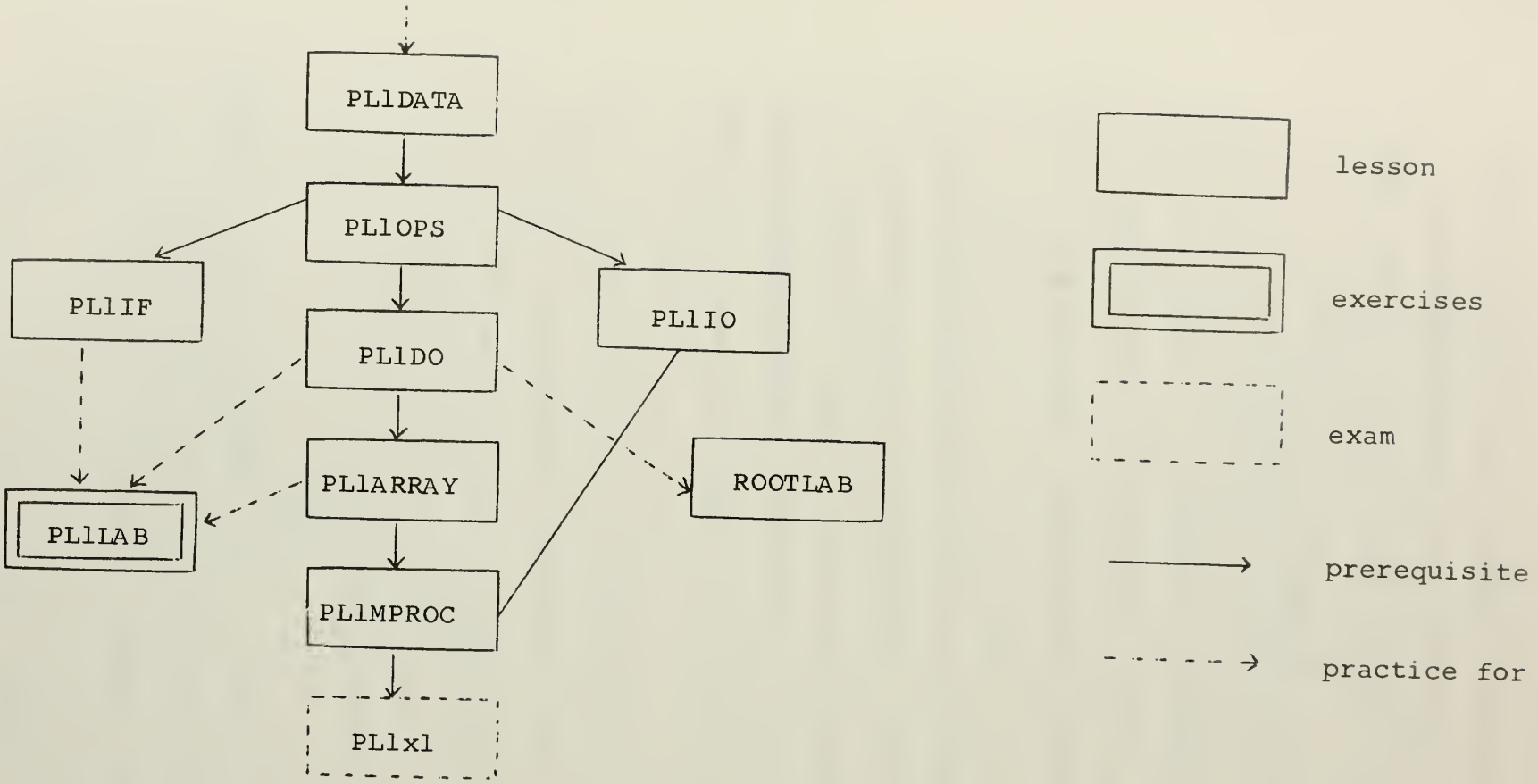


Figure 5. Fragment of the Course Outline

such requests, is largely dependent on the type of data stored in the data bases and the various facts which can be retrieved from them.

The words of a sentence are not symbols having a unique meaning, but they are used and interpreted differently according to the context they are in. The parts denoted by W and S are not always semantically independent. In this sentence:

"At what time does flight AA-57 leave Chicago?" [14]

the word "leave" provides additional information for the W part, which could otherwise have been the same if the person had asked for an arrival time. It is a common observation that any word in a sentence is revealed by its context and is itself part of the context. All words within a given context interact with one another.

These trivial observations lead us to our method which is based on the interaction of words and simulates the thought process of a person reading a context dependent sentence. Let us then describe this thought process.

When we read a sentence, we anticipate naturally the meaning of context dependent words and we memorize automatically the place in the sentence where it occurred. We take the most plausible interpretation of the word and we keep it as long as the succeeding sequence of words does not undermine our confidence in our previous guessing. When the evidence comes that

the interpretation is wrong, we substitute for it another one by changing the meaning of previous context dependent words.

The process is in some respect similar to the traversal of a graph where one node can lead to several others with different probabilities. We simulate this thought process by means of a non-deterministic automaton which identifies the role of each element of a sentence and produces a formal representation reflecting the grammatical structure of the sentence. We found this model appropriate for our application but it would certainly be unsufficient for more general situations. As any automaton, it has input and output symbols and states:

> The input symbols are English words. They represent a rational classification of words according to the meaning they have in particular contexts.
>
> The output symbols are functions. A request is mapped into a nested composition of these functions. The composition reflects the original structure of the sentence.
>
> The states represent a partial degree of understanding and context, which allows the system to generate appropriate comments to an unnacceptable request by looking at the sequences of states and output symbols which have been produced. The state diagram represents the syntax of acceptable requests. It

results that syntactic and semantic analysis are not two separate phases but are merged together. This adds a great deal of flexibility and power to the method. Since this automaton is non-deterministic, each state can lead to several others and each transition has a level of confidence. This level indicates how much confidence the automaton gives to a transition.

We will describe in more detail this automaton and its operation in Chapter 3.

2.5 The Intermediate Representation of Requests

The intermediate representation is an interface between the users and the retrieving components. Its nature is particularly important since it influences strongly the performance of the system. There are as many different intermediate languages as there are such systems. At one extreme they are based on extensions of the predicate calculus; at the other extreme, they take the form of an n-tuple. When we designed the intermediate language, we wanted it to have the following properties:

- It should be able to represent the largest possible number of relevant requests. One way to achieve this goal is to make the language complete. However, no program has so far been able to map all relevant requests into a corresponding formal representation;

completeness in a practical sense cannot be achieved. To remedy this limitation, the request processor in general does not answer strictly the formal request, but expands it to encompass a broader one.

- The translation process should be as simple as possible.
- The formal representation of the request should be such that it permits fast and efficient processing by the retrieving components.

We have chosen to represent the original request of a user by a nesting of functions: f(g(h(x))), where the functions are the output symbols of the automaton. To each function corresponds a set of processing routines. Once the sentence is translated, the request processor processes it from the innermost function to the outermost by calling the appropriate retrieval routines.

We considered the library of lessons as a hyperspace whose dimensions depend on the maximal number of characteristics we attached to any lessons. Some functions return a set of elements from the hyperspace while others return specified characteristics of the elements of this set. The functions may be partitioned into two groups: W' and S'.

The W and S parts of an English sentence (see 2.4) are translated into a composition of functions of W' and S'.

Functions of W':

The twelve functions of W' have as argument a set of lessons. The argument set is defined inside the hyperspace by a function of S' or a composition of functions of W' and S'.

TF : tests if the set is empty or not
AB : returns abstracts of its elements
AN : returns names of author of its elements
NB : returns the size of the set
GR : returns the grade required by the instructor
GO : returns the grade obtained by the student
TS : returns the schedule to achieve relative to the elements of the set
TT : returns the schedule achieved by the student
PQ : returns the set of lessons which are prerequisite to the argument set
SQ : returns the set of lessons which are sequels to the argument set
SI : returns the set of lessons which are similar to the argument set
BE : has two arguments a lesson and a set. Tests if the lesson belongs to the set.

Functions of S':

S' has 2 functions which return a set of lessons depending on the value of their variables.

LN : has two arguments, a course name and a lesson name. It returns the lesson defined by these.
LS : returns the set of lessons which are defined by characteristics other than their names. These characteristics might be a Boolean list of keywords, a type (lesson, exam), a course to which they belong, an author name, a level of difficulty, a sequence specification, a time period, or other specifications, such as whether the lessons have already been taken or not. Any of these characteristics can be specified or negated.

Example of compositions of functions:

- Request "Do you have a lesson on Fortan and numerical analysis?"
- Translation TF (LS (lesson, Fortran & numerical analysis))

- R : "How many prerequisites are required for lesson Fortif in course CS 101?"
- T : NB (PQ(LN(Fortif, CS 101)))

- R : "Should I take Fortdo after Fortif?"
- T : BE(LN(Fortdo); SQ(LN(Fortif)))

- R : "Give me a lesson similar to Fortif."
- T : SI(LN(Fortif))

Looking at these few examples, we see that the intermediate language has the desired properties mentioned previously. The formal representation reflects the structure of the original sentence and makes the translation process easier. The English request is understood as a statement which defines the subset of a set or asks about its characteristics. Hence it prepares and facilitates the task of the request processor.

2.6 The Request Processor

As we mentioned earlier, the implementation of this module has been left as a subsequent task. Let us give here a brief description of its basic functions: It is clear from what precedes that the complexity of the processing varies from one request to another. This processing is, of course, considerably facilitated by the intermediate representation, and let us outline here that when we chose the representation for the formalization of the sentences, we tried to balance the difficulties of the translation process into this representation versus the efficiency of the retrieving process. This efficiency is certainly essential for the overall performance of our system.

The first step taken by this component is to analyze the interpreted request and decide which actions to perform and in which order. This decision process is based on the information handed out by the translator (i.e., the nesting of functions and the value of variables). The following logical steps are then performed at a lower level and depend to some extent on partial results.

A few examples will best describe the functions of this component:

- The request: "What is lesson Fortif about?" is interpreted as AB(LN(Fortif)). It necessitates one access to the record of lesson Fortif in the "Catalog" and the retrieval of its abstract.

- The request: "What are the lessons recommended next after Fortif in course CS 101?" is represented as: SQ(LN(Fortif, CS 101)). The processing is of similar complexity. It consists of an examination of the various links in the "course outline" relating lesson Fortif to the other lessons of course CS 101. A set of lessons (the sequels of lesson Fortif in CS 101) is then determined from this analysis.

- The request: "How many Fortran lessons are there in CS 101?" is interpreted as: NB(LS(lesson, Fortran, CS 101)). The request processor first determines from the "course outline" the set of lessons of

course CS 101, then it determines from the "Catalog" which elements of this set have "Fortran" as a keyword. The ordering in which the searching of the data bases is done is important in regard to the performance achieved. For this request, the "course outline" is accessed first because a course consists typically of less than 20 lessons while the "Catalog" will contain a larger number of lessons having "Fortran" as a keyword.

The processing of requests, however, will not always be as simple as described above. The section (2.3) on the "concept space," shows in particular the ability of the request processor to expand the user's original request to a slightly different field of discourse, for which it has enough information to provide an answer.

3. THE TRANSLATOR

3.1 General Description

To understand adequately the features and capabilities of this system it is necessary first to describe how a request is processed. The interpretation of a sentence consists essentially of finding an acceptable computation in the space of all the computations which can be performed by the non-deterministic automaton.

Let R : $w_1\ w_2 \ldots w_i \ldots w_n$ be the request and L_i denote the number of characters in the substring $w_i \ldots w_n$. w_i and w_{i+1} are words separated by one or more blank characters.

The translator processes the request from left to right, with possibly several backups by the following algorithm.

step 1 Initialize i to 0 and the current state of the automaton to its initial state.

step 2 Increment i by 1.

step 3 if $L_i = 0$ and the automaton is an accepting state, the request is processed.
if $L_i = 0$ and the automaton is not in an accepting state, go to step 6.
if $L_i \neq 0$ try to match a substring starting with w_i with words in the vocabulary data bases.

step 4 If match unsuccessful go to step 2, otherwise get next transition from the automaton. (This transition is determined by the current state and the input symbol associated to the substring just matched.)

step 5 If the output symbol of the transition indicates that a comment should be generated go to step 6, otherwise take the action indicated by the transition, modify accordingly the value of i and go to step 3.

step 6 Generate appropriate comment.

The treatment of a request, as described by this algorithm, is visualized on the screen by the underlining of words as they are processed. When a transition in step 5 leads to the dead state, the value of i is decremented and the processing starts again from the last previous word where there was nondeterminism. The rightmost part of the line is erased accordingly. This visualization has the main advantage of keeping the attention of the user and showing him at any time the stage of the processing of his request (although this processing is fast: 3 seconds on the average).

We will describe steps 4, 5 and 6 in more detail in the following pages.

3.2 The Vocabulary Data Bases

The words w_i, which are recognized during the processing of a request, are of different types:

- "grammatical" words (nouns, pronouns, verbs, auxiliary, adjectives, ...)
- keywords specific to computer sciences
- lesson names
- author names
- course names

When a word of the request is matched against one of these words, it serves as an input symbol to the automaton. The correspondence between the word and the input symbol depends on the type of the word:

- For grammatical words, it has been found useful to have state-dependent input symbols (i.e., the symbol representing a word depends on the current state of the automation). This scheme makes it necessary to store with each grammatical word, a list of input symbols (as many as the number of states). This feature permits the reduction of the size of the automaton and provides more flexibility in its design. (See Figure 6.)
- For the other types of words, the input symbol representing a word is independent of the current state of the automaton. Each type of word is always associated to the same input symbol. This allows a substantial saving in memory space since it is not necessary to store additional information with each word.

The matching of the words of the requests against the words of the dictionary is done by binary search. A hashing technique would have been unfeasible because it is often necessary to match phrases (consisting of several words) in the request in one comparison (for example "numerical

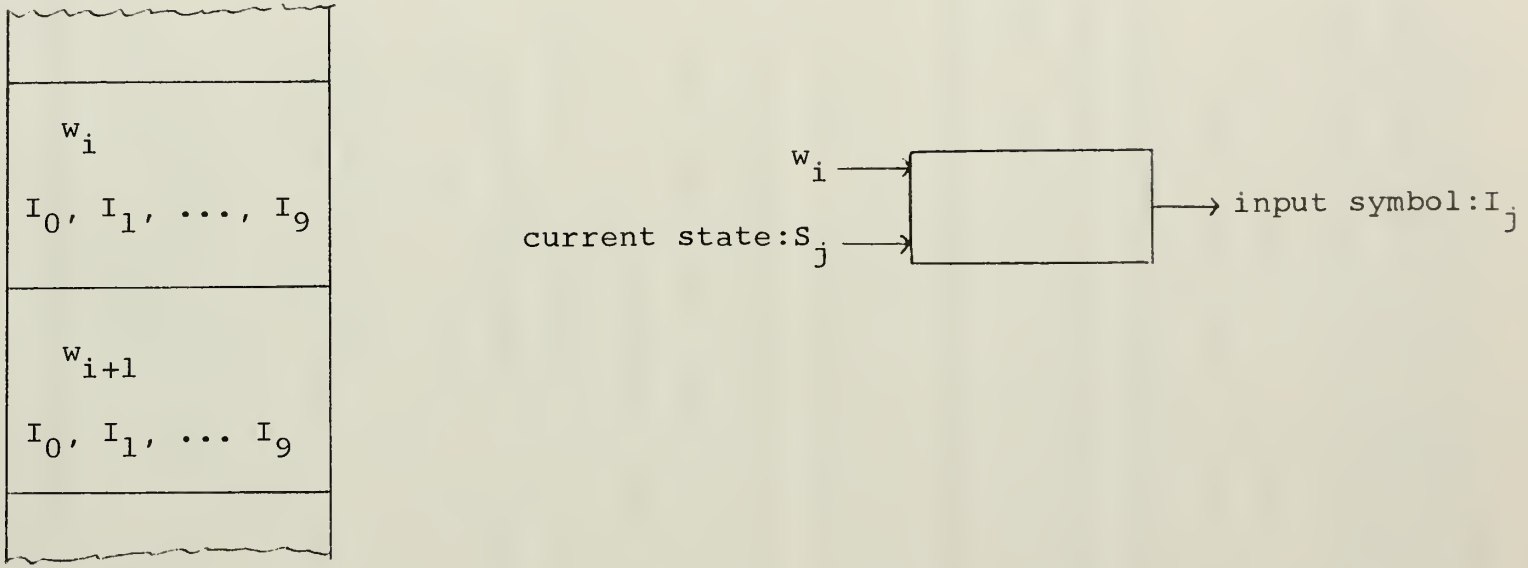


Figure 6. The "Grammatical" Words

analysis," "numerical integration," or "programming," "programming language," "programming technique" ...). The dictionary of words consists of several data bases in which the words are ordered alphabetically. A word is assigned to a data base according to its type and its length. The data bases having the longest "words" are searched first. This allows to distinguish phrases of different lengths where one is the beginning of another: ("information," "information theory," ...).

The dictionary consists of the following data bases:

- 2 data bases for "grammatical words" holding words respectively up to 9 and 19 characters long
- 4 data bases for computer science keywords holding words respectively up to 9, 19, 29, 39 characters long
- 1 data base for lesson names
- 1 data base for author names
- 1 data base for course names

A user can consult any of these last seven data bases before formulating a request. It is important that he uses keywords or names stored in the dictionary since otherwise they are ignored during the processing of the request. The area of memory reserved for the dictionary is large enough to store 1400 keywords of 29 characters each, 200 lessons and author names.

Editing facilities are provided to update easily and rapidly the dictionary. This operation occurs whenever an instructor adds a new lesson to the library or when a new course is created. It is similar to adding a new node to a graph. First the node is defined, then relations with other nodes are set.

3.3 The Automaton

The heart of the translator is a nondeterministic automaton where:

- The input symbols represent categories of words, partitioned according to a rational classification scheme which allows a substantial saving in memory space at the expense of a small increase in computational complexity. Another interesting feature of this scheme is its inherent flexibility: grammatical words can be added to the vocabulary data bases without modifications of the automaton transitions. Let us outline the basic characteristic of the classification:

  Words of the same category belong to the same grammatical class (i.e., nouns, verbs, prepositions, ...) and are:

  - either context dependent synonyms
    ("abstract," "content," "topic," ...)
    ("from," "since," ...)
  - or words which according to a particular context, have different meaning, but

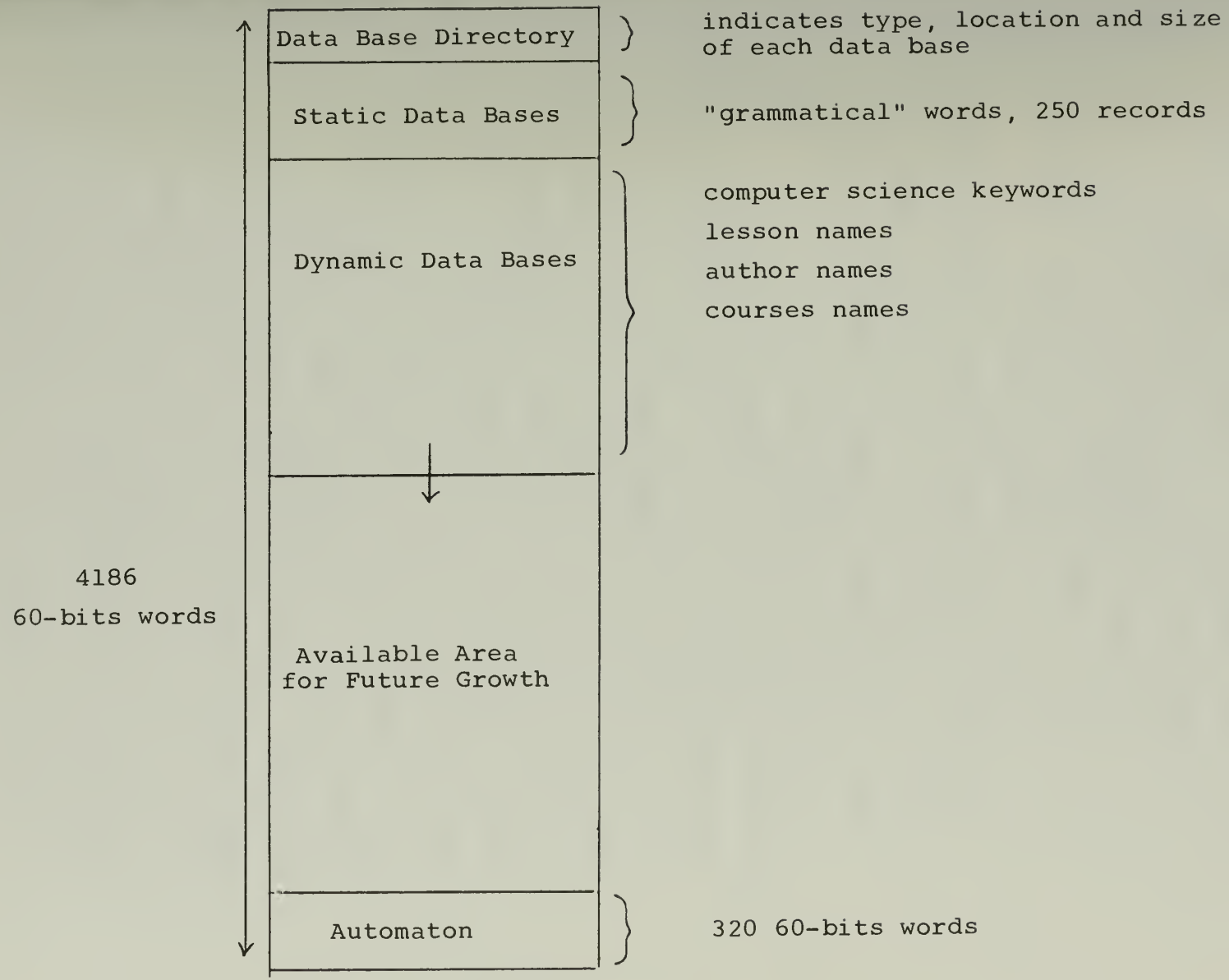


Figure 7. Organization of the Vocabulary Data Bases

serve the same purpose. This is the case of words which correspond to the different values of a function variable. Let us give some examples: ("lesson," "exam," "test") define the type of a library "lesson" ("elementary," "advanced," ...) define a level of difficulty, ("January," "Monday," "today," ...) define a date.

The semantic distinction between these words is made by a routine which also sets the value of the corresponding function variable or flags according to the case.

The automaton has 26 input symbols.

- The states reflect a partial degree of understanding and the state diagram (see Figure 8) reflects the syntax of the accepted requests. The automaton has 11 states including a dead state and 6 accepting states. Let us give a brief description of the state diagram: s_0 is the initial state; s_1, s_2 or s_3 is the current state when the part of the request telling what the user wants to know has been processed. Transitions occur from s_0 to one of these states depending on what the user asks for and the syntax of the beginning of the request:

  "grade required for ..." leads from s_0 to s_1

  "my grade in ..." leads from s_0 to s_3

  "is pℓlif a sequel ..." leads from s_0 to s_2

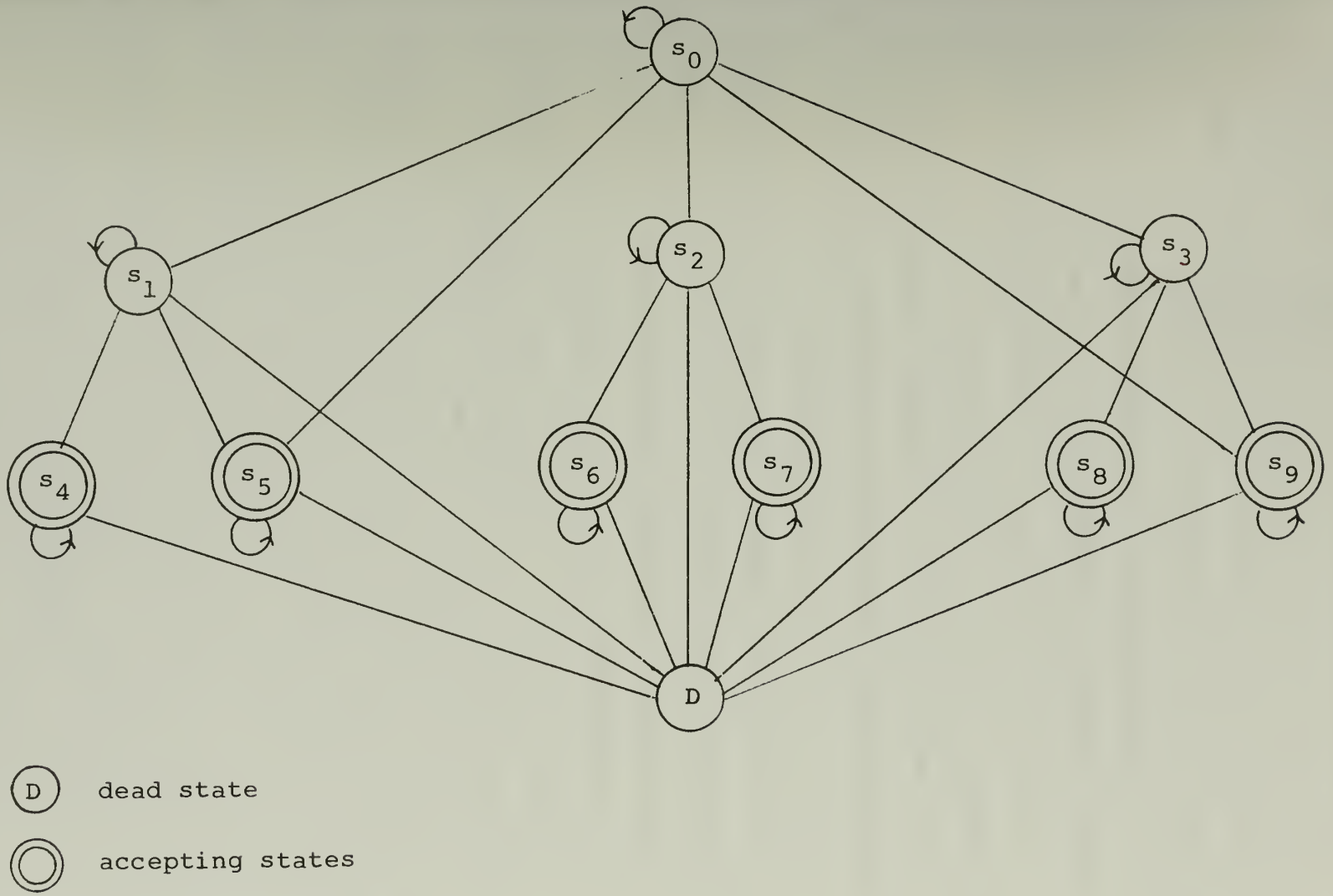


Figure 8. The State Diagram of the Non-deterministic Automaton

"grade I got in ..."	leads first for s_0 to s_1 - a dead state changes then the transition from s_0 to s_3

A transition occurs to one of the remaining states when a specification defining the domain of interest of the user is processed. Several backups may then occur.

11 states have been found sufficient to process a large number of requests. By adding more states, request with unusual syntax like:

"about lesson pℓlif, what is its abstract?"

could have been accepted. We did not do it because adding more states would have doubled the size of the automaton and the concept of partial understanding allows the system to detect such structures and tell the user how to phrase his sentence.

A state can lead to up to four states which are tried in a certain order. Moreover each transition has a level of confidence. When this level is low, the partial interpretation which has been produced is scanned and accordingly either the processing goes on or some comments are generated.

- An output symbol indicates:
  - either a function which will appear in the intermediate representation of the request. Different functions can be attached to an

English word, depending on the current state of the automaton and on the next state chosen. This is a basic feature of the approach we have taken. If this function has variables, the input symbols tell which variable is to be set (15 output symbols are used for this purpose).

- or a flag to be set. Several flags have been found useful in memorizing a few contexts and permitting less and faster computations (6 output symbols).
- The type of a comment to be generated when a transition reveals that the request is out of range or too complicated syntactically or semantically (6 output symbols).

3.4 The Generation of Comments

The concept of partial understanding, inherent to our approach, allows the generation of specific comments whenever a request is outside the accepted set. Appropriate comments prevent the user from becoming frustrated and help him to use the system in a better way.

All the comments are generated by a single routine which receives as input the type of the comment to be displayed and assembles the resulting comment from a number of stored messages which can be used in different circumstances. During this operation, the routine may use the sequence of output

symbols or states which has been produced to output a specific statement. It also keeps track of the number of previous comments and uses it to reinforce the comment generated.

Its operation is summarized by the schema in Figure 9.

Example of Comments

R: What do I request?
C: Please look at the purpose of the Guide and examples of requests.

R: What grades did I get?
C: I understand you want to know your performance. Your request is not specific enough.

R: What grades should I get?
C: Your request is not specific enough.

R: What should I do until the fifth of January?
C: Sorry, I cannot answer this request. Do not use the word which is underlined.

R: Show me the abstract of pl1do and the name of its author.
C: Please phrase simpler requests.

R: What lessons did I study since the 10?
C: When you specify a date by a number, it must be preceded by the name of a day or of a month.

R: Give me a lesson about Fortran or pl1 and numerical analysis.
C: Indicate only a conjunction or a disjunction of computer science keywords.

R: Is pl1do a better lesson than Fortdo?
C: Sorry, I cannot answer this request. I don't make such comparisons.

When successive requests are refused, additional comments are given each time:

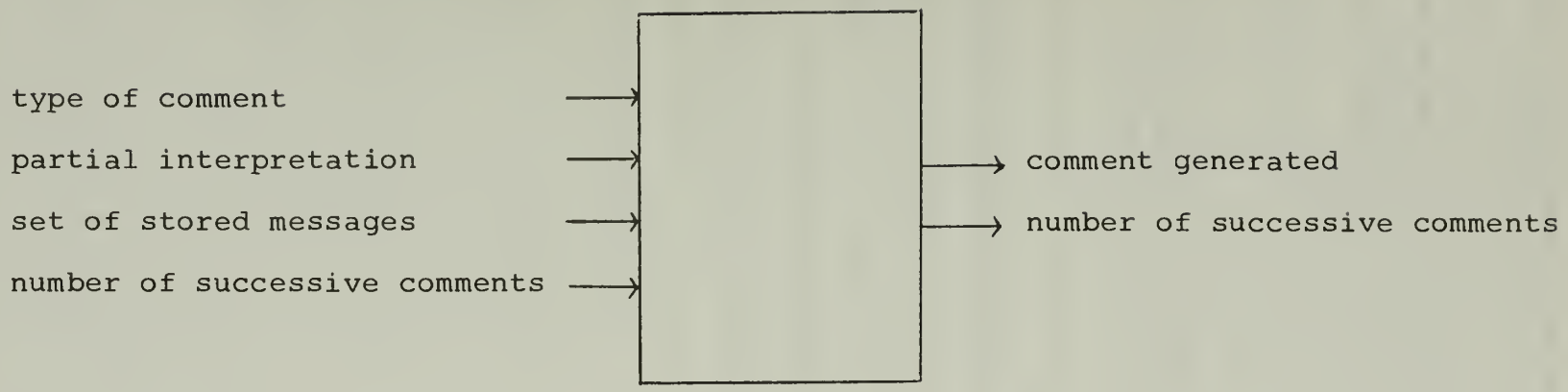


Figure 9. The Generation of Comments

R1: Lesson Somaga, who wrote it?
C: Please phrase simpler request.

R2: Somaga, who wrote it?
C: Your request is not yet acceptable. Phrase your request differently. Indicate first what you want to know then specify which lesson.

R3: Who are the authors of lessons Somaga and pɬlif?

C: Again! You really give me a hard time. Please phrase simpler requests! You have indicated too many lessons.

3.5 Theoretical Considerations

We have given in the previous sections the description of a cybernetic system which processes information by means of a non-deterministic automaton. We present in this section a formal description of this automaton and justify the choice of this model rather than a deterministic or stochastic one.

We can characterize our model by:

$$ND = [I, O, S, h]$$

where I: denotes the set of input symbols (i). For reason of simplicity we can identify here I with the set of English words or phrases stored in the dictionary.

O: denotes the set of output symbols (o). The intermediate representation of a request consists of a composition of elements of this set.

S: denotes the set of states (s).

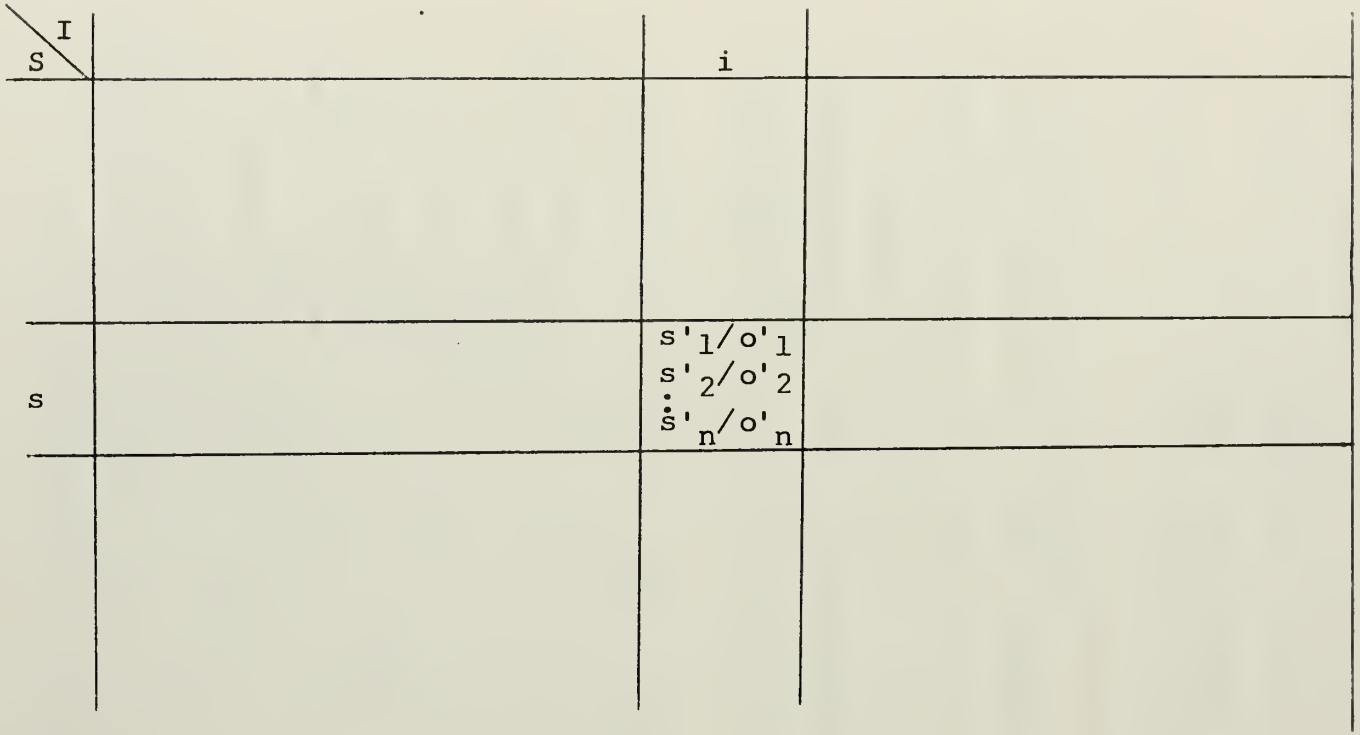
h(i,s): denotes the nonempty set of ordered pairs (o,s') from which the output symbol (o) and the next state (s') can be chosen in the situation (i,s).

The need for such a model has been explained previously. We can justify it again shortly by establishing a parallel between the multiple interpretations of certain words or phrases in a sentence in regard to its overall meaning and the multiple possibilities of ordered pairs (o,s') which can result from a situation (i,s). We implemented a non-deterministic Mealy automaton (see Figure 10) rather than a Moore model.

Let us now justify the choice of this model rather than a deterministic or stochastic one.

Deterministic automaton. In a deterministic machine, the set h(i,s) contains only one pair (o,s'), that is, the output symbol (o) and the next state (s') are uniquely determined in a given situation (i,s). The fact that words or phrases in a sentence may have different interpretations excludes this model.

Stochastic automaton. This category of machines has been found useful for modelling certain forms of behavior and in particular to clarify the notion of intelligent behavior. With this model, a pair (o,s') is chosen from the set h(i,s) with a probability which depends only on the situation (i,s). This property makes the model unfeasible for our application since we must choose the pair (o,s') according to the past history of the situation (i,s). In our non-deterministic model the pairs (o,s') are taken from the ordered set h(i,s) according to the order of the occurrence of the situation (i,s).



$h(i,s) = (o'_1,s'_1), (o'_2,s'_2), \ldots (o'_n,s'_n)$

Figure 10. Non-deterministic Mealy Automaton

4. PERFORMANCE AND APPLICABILITY OF THE TRANSLATION METHOD

4.1 Performance

The size of the accepted set is one of the most important criteria when judging the usefulness and performance of a natural language interface between an information system and its users. Another criterion is the kind of answer given when a request is slightly outside the accepted set. A system which simply fails in the interpretation and rejects the request without further explanation raises doubts about the usefulness of such an interaction and is easily subject to criticism if its set of accepted requests is not large enough.

We defined this set iteratively. Initially lists of requests by half a dozen students set the first boundaries. Then several months and modifications later, when the system had been implemented, we asked a class of students to test it intensively. About 300 requests were thus generated and although most of the relevant ones were accepted, their analysis revealed that the boundaries had to be expanded by adding more states to the automaton. This iterative approach has been facilitated by keeping the system always as modular as possible. Our vigilant constraints, described previously, have ended the iteration and we can evaluate the system in its final state.

Let us first list a few requests which are accepted:

Sample of requests about lesson names

- Do you have any advanced lessons on pℓ1 or algol written by Smith, that I have not yet taken?
- What are the lessons I took from September 10 until yesterday in cs 101 and which are not about pℓ1 or Fortran?
- Let me see the sequels of the lesson I took last time.
- List all your lessons on list processing which have exercises and are not required in cs 105.
- What lessons can I take after Fortdo?
- Tell me a lesson similar to the last I studied.
- Help! I need practice in pℓ1.

Sample of requests for information intrinsic to lessons

- Abstract of the prerequisites of lesson Fortdo.
- What is the topic of my next lesson?
- Who wrote the lessons I took today?
- Is Fortdo similar to lesson pℓ1do?

Sample of requests about student performance

- Tell me my grade in my last exam.
- Which grade did I get in lessons taken since August?
- How much time did I spend in somaga?
- How many lessons did I take so far in cs 105?
- How many hours have I spent on pℓ1 lessons?

Sample of requests about lesson requirements

- What is the grade required in the exam pℓ1X1?
- When should I take my next exam in cs 105?
- What are the prerequisites of lesson Fortdo?
- Should I learn pℓ1if after pℓ1do?
- Is it necessary to go through lesson Epic in cs 101?
- Can I program in pℓ1 now?
- Should I study pℓ1if before November?

These few examples illustrate the types of requests which are accepted. The system, however, does not process all relevant requests since it is not complete (section 1.4) and its fluency has limitations related to the complexity of the state diagram of the automaton. Nevertheless, experiments have shown that most of the relevant requests phrased by students unfamiliar with the system have been fully interpreted and that the comments have been found useful in helping them at the beginning.

A criterion of importance besides the size of the accepted set is the response time of the system. A time consuming interpretation would have prohibited this kind of interface since the PLATO time sharing system aims at serving users only 1 ms/s on the average, with more time allocated, if needed, for short periods. The interpretation of a request takes about 60 ms of processor time. This seems too much according to the specifications of PLATO, but the Guide is not executed for long (about one minute) as opposed to one hour for a lesson.

The following examples show requests with the time needed for their interpretation. (TP: Time needed by the processor, PRT: PLATO's Response Time.)

- "Abstract of plif."
  PRT: 2 secs TP: 14.9 msecs/sec

- "Let me see a lesson about Fortran and numerical analysis."
  PRT: 4 secs TP: 16.6 msecs/sec

- "Is pldo similar to lesson Fortdo?"
  PRT: 3 secs TP: 27.0 msecs/sec

4.2 Conditions for its Applicability

In this section we will outline how the project has matched our initial goals and the contribution it represents for information systems.

Designers of systems inevitably meet constraints and adjust the scope of their project accordingly. In our case, we could not adopt a "successive approximation" technique. The stringency of the constraints and the goal of the system compelled us to find new techniques to provide a natural language communication with its users, and this constituted a challenging research problem. The performance characteristics described in a previous section are convincing enough to show that the system has fulfilled the initial goal despite the memory space and response time constraints imposed by the PLATO system.

The method we used for the interpretation of requests and their mapping into a search prescription is independent of the domain of discourse and can be employed whenever the

sentences remain within a sufficiently restricted sphere. The body of requests for information related to a restricted realm of knowledge generally satisfy this condition and the structure of these sentences has properties which facilitate their interpretation. We list below a few instances where this technique could be used profitably and ultimately we show how the method could simplify an already existing system which uses a different approach for the understanding of requests.

4.3 Feasible Applications

This method can be applied with benefit to similar question-answering systems, which deal with occasional users and cannot afford a time-consuming communication with the outside world by large programs. This situation exists for many real time systems where a short response time is desired. The method can also be useful for:

- Fully automated document retrieval systems which only interpret requests as a list of keywords to be matched against those of stored documents. These systems do not retrieve facts although it would certainly improve their overall performance. A system like Smart [10] does not interpret negated keyword.
- Information systems where the user requests must first be mapped into a search prescription by a non-mechanized operation.

4.4 Application to an Existing System

As an example, we will now apply our method to a domain of discourse of an existing similar system and show how our approach makes the interpretation of sentences easier.

W. Wood [14] has developed a system, having the Official Airline Guide as a data base, for performing semantic interpretations of English questions like:

- "How many flights go from Boston to Chicago?"
- "Does AA-57 go from Boston to Chicago?"
- "Name 5 flights that go from Boston to Chicago."

The interpretation of requests is conceptually divided into two independent phrases, syntactic analysis and semantic analysis as shown in Figure 11.

A syntactic analyzer first maps a request into a structure which explicitly represents the grammatical relationships among the words of the sentence; this structure is then interpreted by the semantic analyzer with a set of rules of the form: (pattern $\Longrightarrow$ action) where "pattern" is a substructure and "action" its interpretation.

The method consists of interpreting directly substructures and deriving the semantic interpretation of the request from the composition of the interpretations of its substructures. Relations asserted by a sentence are determined by the main verb which plays a fundamental role in the analysis. This system has been written in LISP and implemented on the Harvard Time Sharing system.

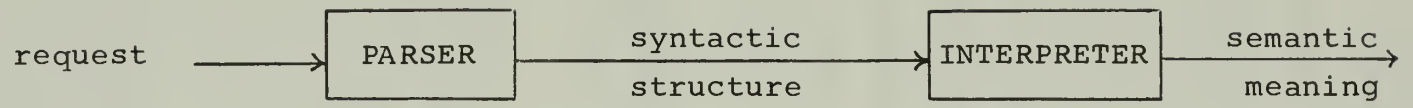


Figure 11. Parsing and Interpretation of a Request

We will now show how our method could have been used and take a representative sample of requests (listed in [14]) processed by this system as the domain of discourse.

4.4.1 Intermediate Language

Using the same terminology as before, a request will be mapped into a composition of functions belonging to W' and S'.

Functions of W':

AL	returns Airline Company name
DT	returns a Departure Time
AT	returns an Arrival Time
NB	returns the size of the set defined by the argument
TF	Test if the set defined by the argument is empty or not. (True False)

Functions of S':

S' has only one function of seven variables:

FS (V1, V2, V3, V4, V5, V6, V7) which consist of flight specifications and can be negated.

V1 : flight quantifier

V2 : flight name

V3 : Airline name

V4 : Town of departure

V5 : Town of arrival

V7 : departure time DT

V8 : Arrival time AT

with	V1	number | comp.number
	comp	< | > | ≤ | ≥
	V7	date | prep.date
	V8	date | prep.date
	prep	before | after

FS return the set of flights which verify the specifications.

4.4.2 State Diagram

The state diagram (see Figure 12) is simpler than the one of the Guide because the meaning of words of the requests is not so context dependent. However, the number of states and functions of W' or S' would increase if we want to process more complex requests than the ones given as a sample by Woods in [14].

4.4.3 Translation of a Sample of Requests

The sample of requests listed in [14] would be interpreted as follows:

- Doesn't American operate flight AA-57?
  TF (FS(1, AA-57, American, _, _, _, _))
- Isn't AA-57 an American Airlines flight?
  TF (FS(1, AA-57, American, _, _, _, _))
- Does American have a flight which goes from Boston to Chicago?
  TF (FS(1, _, American, Boston, Chicago, _, _))
- Does American have a flight which doesn't go from Boston to Chicago?
  TF (FS(1, _, American, ¬Boston, Chicago, _, _))
- What American Airlines flight goes from Boston to Chicago?
  FS (1, _, American, Boston, Chicago, _, _)
- What American Airlines flights go from Boston to Chicago?
  FS (_, _, American, Boston, Chicago, _, _)

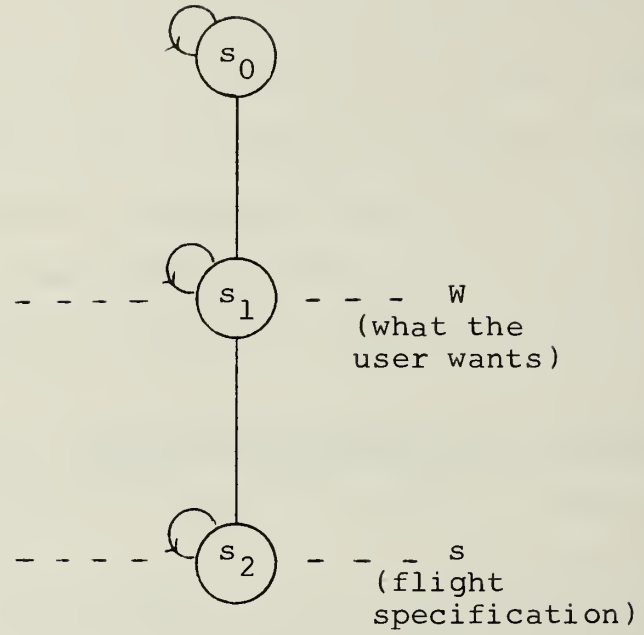


Figure 12. State Diagram for Airline Guide [14]

- What is the departure time of AA-57 from Boston?
  DT (FS(_, AA-57, _, Boston, _, _, _))

- What is the departure time from Boston of every American Airlines flight that goes from Boston to Chicago?
  DT (FS(_, _, American, Boston, Chicago, _, _))

- What American Airlines flights arrive in Chicago from Boston before 1.00 p.m.?
  FS (_, _, American, Boston, Chicago, _, < 1.00 p.m.)

- How many flights that go from Boston to Chicago does American Airlines operate?
  NB (FS(_, _, American, Boston, Chicago, _, _))

- What is the number of flights from Boston to Chicago?
  NB (FS(_, _, _, Boston, Chicago, _, _))

LIST OF REFERENCES

[1] Alpert, D., Bitzer, D. L., "Advances in Computer Based Education," Science, 167 (1970), pp. 1582-1590.

[2] Bobrow, D. G., "Natural Communications with Computers," Bolt-Beranek Newman (1967).

[3] _____, "Problems in Natural Language Communications with Computers," Bolt-Beranek Newman (1967).

[4] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Comm. ACM, 13, (1970), pp. 377-387.

[5] Craig, J. A., Berezner, S. C., Carney, H. C. and Longyear, C. R., "DEACON: Direct English Access and Control," AFIPS Conference Proceedings, Vol. 29, The Thompson Book Company, pp. 365-380.

[6] Engles, R. W., "A Tutorial on Data-base Organization," Annual Review in Automatic Programming, 7 (1972), Pergamon Press, pp. 1-64.

[7] Green, B. F., Wolf, A. K., Chomsky, C. and Laughery, K., "BASEBALL: An Automatic Question Answerer," in E. Feigenbaum and J. Feldman (Eds.) Computers and Thought, New York: McGraw-Hill (1963).

[8] Nievergelt, J., Reingold, E. M., Wilcox, T. R., Watanabe, D. S., Friedman, H. G., Montanelli, R. G. and Pradels, J., "ACSES: An Automated Computer Science Education System," to appear in Proc. 1974 National Computer Conference, AFIPS Conference Proceedings, Vol. 43, AFIPS Prsss.

[9] Quillian, M. R., "The Teachable Language Comprehender " Comm. ACM, 12 (1969), pp. 459-476.

[10] Salton, G., "The SMART Retrieval System," Prentice-Hall, Inc.,(1971).

[11] Sherwood, B., et al., "aids," PLATO IV Lesson, Computer-based Education Research Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois.

[12] Simmons, R. F., "Natural Language Question-Answering Systems," Comm. ACM, 13 (1970), pp. 15-30.

[13] Winograd, T., "Understanding Natural Language," Cognitive Psychology, 3 (1972), pp. 1-191.

[14] Woods, W. A., "Procedural Semantics for a Question Answering Machine," AFIPS Conference Proceedings, Vol. 33, Part 1, The Thompson Book Company (1968), pp. 457-471.

[15] _____, "The Lunar Sciences Natural Language Information System," Bolt-Beranek Newman (1972).

[16] _____, "Transition Network Grammars for Natural Language Analysis," Comm. ACM, 13 (1970), pp. 591-606.

APPENDIX A

DETAILED STRUCTURE OF THE TRANSLATOR

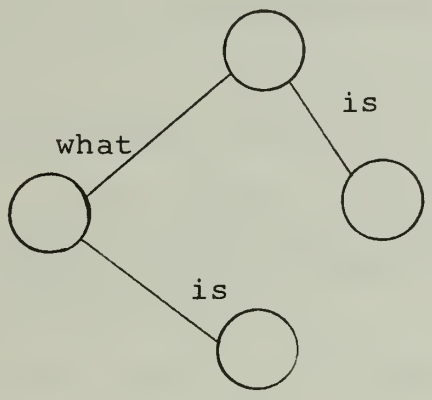
This appendix contains a more detailed description of the translator. First we list the grammatical words which can be recognized during the processing of a request, then we describe which automaton states are involved in the interpretation of various requests, and give a sample of words which cause a transition from one state to another.

1. List of Grammatical Words

We called "grammatical words" the words which need to be recognized in the sentence and which are not specific to the domain of discourse, like computer science keywords, lesson names, author names or course names. These grammatical words are the most important ones for determining the interpretation of the request, and most state transitions are caused by them. The dictionary contains about 250 of them.

Before listing these words, let us point out that the dictionary of English words has some idiosyncrasies due to our concern of saving memory space and having fast interpretations. The dictionary contains a few expressions consisting of two or more words which are considered as individual entities and hence recognized in "one operation." Let us

take an example to show why this has been done. A sentence beginning by "what is" will probably yield a different interpretation than a sentence beginning by "is." If every single word is recognized, then it is necessary to have a large number of states and assign different transitions to each word.



This approach complicates unnecessarily the design of the automaton and is moreover costly in terms of memory space required for both the automaton and the dictionary. It would also go against our design principles of having each state representing a partial degree of understanding. Instead we included in this dictionary expressions of words like: "what is," "how do," "next after," "last time," "my grade," ... . This solution is inevitably pragmatic, but contributes greatly to a substantial saving in memory space and to a faster interpretation of requests.

The following words may partially indicate facts a user is interested in, depending on where each individual word occurs in a sentence and also on the preceding and succeeding parts of the sentence:

(abstract, description, outline, subject, summary, topic, information)

(author, who)

(how many, how much, number)

(grade, level, performance, standing)

(hour, how long, schedule, time, when)

The following words may define a set of lessons in terms of another one:

(prerequisite, requirement; continuation, sequel)

(like, same, similar, related to)

The following words may define some attributes of a "lesson":

- its type: (lesson, exam, test)
- its level of difficulty: (advanced, difficult, easy, elementary, hard, introductory, simple)
- its purpose: (exercise, practice, practicing, theoretical, theory, to program)
- a date or a period: (January → December, Monday → Sunday, yesterday, now, today, tomorrow)

The following words indicate the verbs and their tenses, which are recognized:

(are, do, give, go through, have, is, learn, list, need, show, study, take, teach, tell, want, will)

(did, had, gone through, learned, saw, seen, shown, studied, taken, took, was, went through, were)

Prepositions:

(about, after, and, before, between, but, by, during, except, first, from, following, in, last, next, not, no, on, or, previous, preceding, since, succeeding, to, until, without)

The groups of words considered as single entities:

(how are, how bad, how can, how do, how good, how have, how is, how well)

(my grade, my level, my performance, my standing)

(what are, what did, what do, what have, what is, what will, what should, what would, what else)

Finally there are the words which indicate independently of the state of the automaton that the request has a high probability to be out of range, or words which cannot be interpreted by the translator, due to its limitations:

(why, reason, advice, improve / this, these, those its, their / week, month, day, semester, year / one, two, three, ... / second, third, ... / 'comparatives and superlatives')

When these words are recognized, comments are displayed on the screen of the user, expressing the limitation of the system.

Besides the words listed previously, numbers are also recognized and accepted only in the context of a date or period specification. Otherwise, appropriate comments are generated.

As we mentioned earlier, this classification of words is superficial. Words within one group can be interpreted differently depending on the context, and they can cause different state transitions. However, this partition is useful to give an idea of the words stored in the dictionary and will be helpful to give an overview of the mechanism of the automaton.

2. The State Transitions

The sentences accepted by the translator can be divided into three major categories which correspond to a partition of the states of the automaton into three subsets:

- The first category consists of requests which ask for information specific to some lessons or for information about the performance or the schedule to achieve by a student among the various lessons of a course he is registered in.
- The second category consists of requests which ask for the performance or schedule achieved by a student in the various lessons he has already taken.
- The third category consists of requests about the existence of a set of lessons or about the inclusion of a lesson defined by its name into another set of lessons.

2.1 Interpretations of the Requests of the First Category

The interpretation is done by the following subset of states:

state s_0: initial state

state s_1: The current state is s_1 when the automaton has made a guess about what the user wants to know.

state s_5: The current state is s_5 when the automaton has recognized or is waiting for words which define a set of lessons by some attributes different from a name. It is in this state and during transitions from another state that the variables of function (LS) are set.

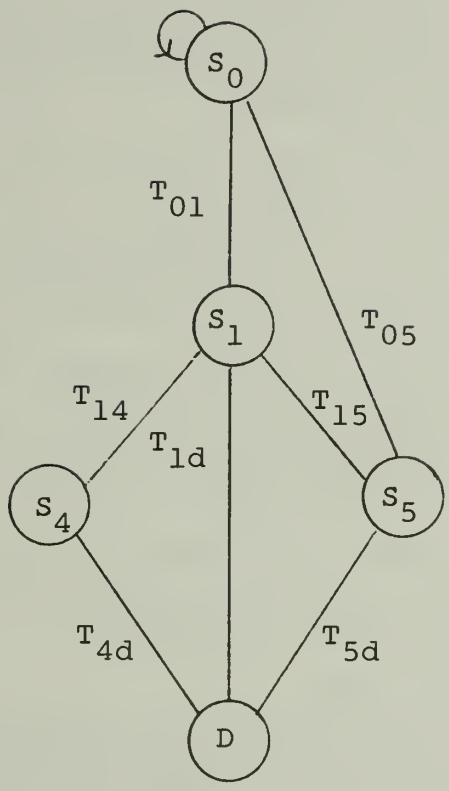


Figure 13. Subset of States

state s_4: The current state is s_4 when a lesson is defined by its name in the request and when there has been an interpretation about what the user wants to know about it. (The variables of function (LN) are set.)

dead state: A transition to the dead state occurs from either s_1, s_5, or s_4 when some word in the sentence indicate that the interpretation of previous word is incorrect.

We list below a sample of words which cause transitions. F_{ij} and S_{ij} denote the sets of words for which T_{ij} is the first transition or a subsequent one (second, third or fourth), respectively.

$\underline{T_{01}}$: F_{01}: ("Abstract," "author," "time" "prerequisite," "similar," "number," "grade")

S_{01}: ("what is," "want," "lesson," "take")

$\underline{T_{05}}$: F_{05}: ("lesson," "take")

S_{0s}: no words

$\underline{T_{15}}$: F_{15}: (words which can indicate a value for the variables of function LS, "on," "by," "about," ...)

S_{15}: no words

$\underline{T_{14}}$: F_{14}: (a lesson name)

S_{14}: (a course name)

$\underline{T_{1d}}$, $\underline{T_{4d}}$, $\underline{T_{sd}}$: words which indicate that the request belongs to the second category: ("obtained," ...) or which modify a previous interpretation:

<u>example of interpretations</u>:

"<u>abstract</u> of a <u>lesson</u> <u>on</u> <u>numerical analysis</u>."

S_0: S_1 S_5 S_5 S_s

AB (LS(lesson, numerical analysis)

"What is the abstract of lesson plif?"

S_0: S_0 : S_1 S_5 D

S_0 : S_1 S_1 S_4

AB(LN(plif))

"What is plif about?"

S_0: S_0 D

S_s D

S_1 S_4 S_4

AB(LN(plif))

"What lessons should I take before plif in cs 101?"

S_0: S_5 S_5 S_5 S_5 D

S_0 S_5 S_5 S_5 D

S_0 S_5 S_0 S_1 S_4 S_4 S_4

PQ(LN(plif, cs 101))

2.2 Interpretations of the Requests of the Second Category

The interpretation is done by the following subset of states:

- The significance of the states S_2, S_6, S_7 is the same as the one of states S_1, S_4, S_5 described previously. However, the set of words causing a transition to the dead state is different.

  T_{02}: F_{02}: ("my grade," "my performance," ...)

  S_{02}: ("grade," "time")

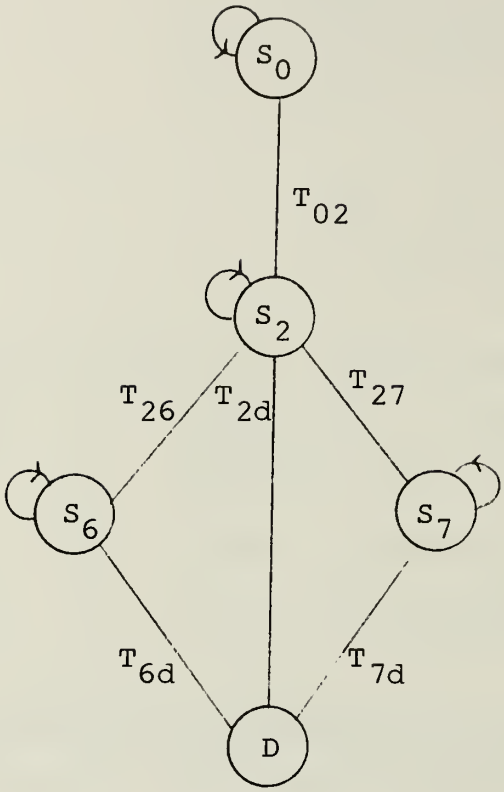


Figure 14. Subset of States

T_{26}, T_{27}: Words which cause transitions from S_2 to S_6 or S_7 are respectively the same as those which cause a transition from S_1 to S_4 or S_5.

T_{2d}, T_{6d}, T_{7d}: These transitions do not occur for words like ("obtained," "got") as they did previously from states S_1, S_4, S_5.

example of interpretations:

"Tell me my grade in the last exam."

S_0: S_0 S_2 S_7 S_7

GO(LS(exam, last))

"Tell me the grade I got in my last exam."

S_0: S_0 S_1 D

S_0 S_2 S_2 S_7 S_7 S_7

GO(LS(exam, last))

"When did I take pℓ1X1?"

S_0: S_1 D

S_2 S_5 D

S_2 S_2 S_6

TT(LN(pℓ1X1))

2.3 Interpretations of the Requests of the Third Category

The interpretation is done by the following subset of states:

These requests begin typically by words like ("is," "any," "have"). The first guess by the automaton is that the request is about the existence of a set of lessons defined by some

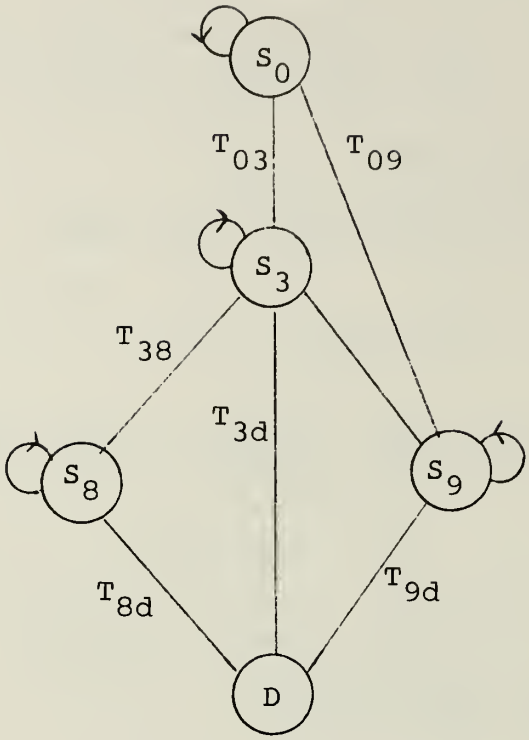


Figure 15. Subset of States

attributes. Hence the first transition which occurs in T_{09} where S_9 has the same significance as S_5 or S_7.

If subsequent words indicate that this partial interpretation is wrong then the transition T_{03} is attempted. The interpretation anticipated is that the request is about the inclusion of a lesson uniquely defined into a set of lessons. The significance of states S_8 and S_9 is the same as the one of states S_4 and S_5 or S_6 and S_7.

T_{09}: F_{09}: ("is," "do," "any," ...)

S_{09}: no words

T_{03}: F_{03}: no words

S_{03}: ("is," "do," "any," ...)

T_{39}: F_{39}: (lesson names, "prerequisite," "sequel," "before," "after," ...)

S_{39}: no words

T_{38}: F_{38}: no words

S_{38}: ("prerequisite," "sequel," "before," "after," ...)

example of interpretations:

"Have you a lesson on numerical analysis?"

S_0: S_9 S_9 S_9 S_9

TF(LS(lesson, numerical analysis))

"Have you the abstract of lesson plif?"

S_0: S_9 D

S_3 D

S_0 S_1 S_5 D

S_0 S_1 S_1 S_4

AB(LN(plif))

"Is plif a prerequisite of plldo?"

S_0: S_9 D

S_3 S_9 D

S_3 S_3 S_9 D

S_3 S_3 S_8 S_8

BE(LN(plif), PQ(LN(plldo)))

"Is plif a lesson on conditional statements?"

S_0: S_9 D

S_3 S_9 S_9 S_9 S_9

BE(LN(plif), LS(lesson, conditional statement))

APPENDIX B

THE DATA BASE ORGANIZATION

We have given in section 2.3 a description of the logical structure of the Concept Space, the Catalog, and the Course Outline. We present here a scheme for the implementation of these data bases which allows efficient memory allocation and search operations.

Estimates have shown that, when the system is used, the three data bases can be kept entirely in an area of restricted size (4186 60-bits words) of the Extended Core Storage (ECS), a level of the computer memories hierarchy for which transfers of data with the main memory occur at a fast rate (ECS access time = 5 microseconds, Transfer rate = 600 millions bits/second). These estimates are based on the following figures: 1024 computer science keywords, 200 concepts, 128 lessons and 16 courses.

The implementation scheme is illustrated in Figure 16. Each data base has a directory, relating a record identifier to the storage location of the corresponding record. A hashing function is used to determine the entry in the directory which corresponds to a record identifier. The records are of variable length and have a field called the record descriptor, which

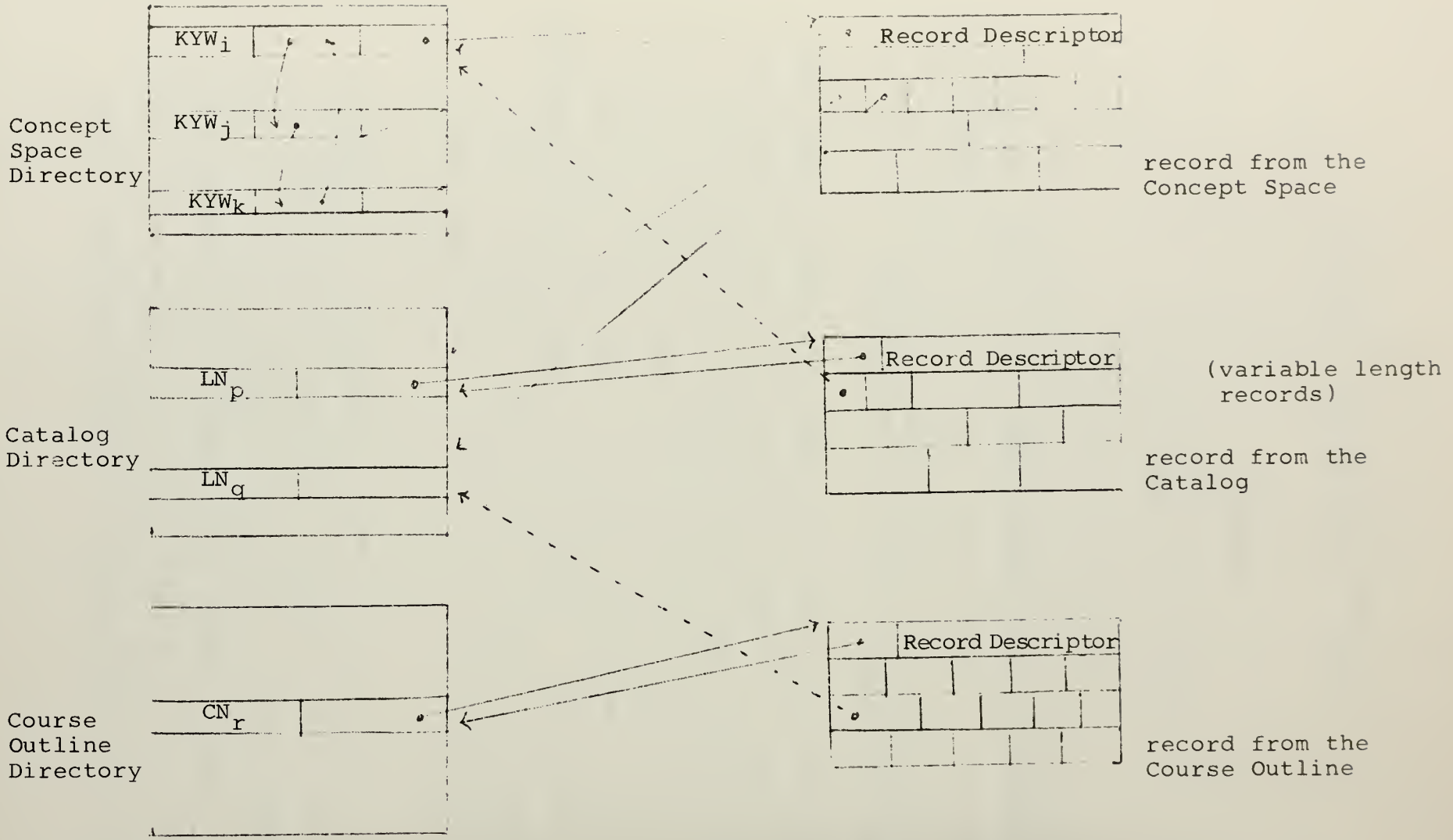


Figure 16. Data Base Organization

describes the internal organization, i.e., the location and the length of the other fields.

Organization of the Concept Space

The directory. A record identifier is a computer science keyword which has been mapped into a fixed length key (see section 2.3). This directory is the largest one since it will contain approximately 1000 entries. Some record identifiers are linked by a ring; they correspond to computer science keywords which represent the same concept. Only one of them gives the location of the record. This scheme implies a linear search over a few elements to access a record but makes the updating of the data bases more efficient, as we will explain later. This ring is also used by the request processor to obtain the list of all the keywords which characterize a concept.

The records. Each record represents a computer science concept and has a field called the record descriptor which describes its internal organization. The information contained in these records has been described in section 2.3. The list of lessons which are relevant to a concept is also added to the corresponding record. Since each entry in the directories has a fixed location, it is a list of pointers rather than symbolic names. These pointers indicate the address of some entries in the Catalog directory. This representation contributes to a substantial saving in memory space (8 bits instead of 60 bits to represent a lesson in the list). Hence

by adding this list to each record, the Concept Space serves also the purpose of the inverted file described in section 2.3.

Organization of the Catalog and Course Outline

The organization is essentially the same as described previously. A record of Catalog has a lesson name as an identifier and contains an information which characterizes a particular lesson. The list of computer science keywords attached to each lesson is represented in a record by a list of pointers indicating some entries in the Concept Space directory. A record of Course Outline has a course name as an identifier and contains an information relative to the requirements and schedule of a course. Lessons are represented also by pointers to some entries in the Catalog directory rather than by symbolic names.

Maintenance

Deletions of records create holes of various sizes. Insertions of new records could then be performed by using the "first fit" or the "best fit" algorithms as described by Knuth, but whatever is the method, holes remain and give to the area reserved for storing the data bases, the appearance of being "checkerboarded." The restricted size of this area prohibits these insertion methods. In our system, updating operations do not occur frequently, but mainly at the end of a semester. Hence the "compacting of the memory," i.e., moving the records until the holes have been coalesced into a single

hole, whenever one or several records have been deleted, is an appropriate solution which allows an efficient use of the area. In each record a pointer is added which indicates the location of the record identifier in the directory. Whenever records are moved this pointer allows to update directly the corresponding entries in the directories. The use of this pointer makes the compacting operation much more efficient than if the updating of the directories was done afterwards, particularly if we consider that the number of entries in the Concept Space directory is about five times the number of the records of this data base.

VITA

The author, Jean Louis Pradels, was born in Chavagnac, France on February 5, 1946. He received a "diplome d' ingenieur" from the Ecole Superieure d' Electricite' in Paris in June 1970 and his Master of Science degree in Computer Science in June 1971 from the University of Illinois. Since September 1972 he has been a research assistant in the Department of Computer Science of the University of Illinois at Urbana-Champaign. He is a member of Sigma XI.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-74-626	2.	3. Recipient's Accession No.

4. Title and Subtitle

THE GUIDE - AN INFORMATION SYSTEM

5. Report Date: March 1974

6.

7. Author(s)

Jean Louis Pradels

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

University of Illinois
Department of Computer Science
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.

GJ 31222

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, D.C.

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

This thesis describes the design and implementation of a conventional information system, called the Guide. The environment in which it must work is sufficiently different from that of other systems which have similar goals, that new techniques had to be devised to make its realization possible. This thesis also shows that these technqieus are applicable in contexts other than ours, and would probably lead to improved performance in a number of information systems based on other approaches.

17. Key Words and Document Analysis. 17a. Descriptors

information system
natural language
computer-assisted-instruction

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page) UNCLASSIFIED	22. Price